//Charlie Ang

//March 8, 2017

//CSC 3430

//Homework 6: Designing and implementing program that uses a Dynamic Programming solution to

//compute the "minimum editing distance" between two strings and shows the operations necessary to

//transform the first string into the second string

//Code borrowed from: http://www.geeksforgeeks.org/dynamic-programming-set-5-edit-distance/

// A Dynamic Programming based C++ program to find minimum

// number operations to convert str1 to str2

#include <string>

#include <iostream>

#include <iomanip>

#include <stack>

using namespace std;

// Utility function to find minimum of three numbers

int min(int x, int y, int z)

{

if (x < y && x < z) return x;

if (y < x && y < z) return y;

else return z;

//return min(min(x, y), z);

}

//Function to print out relevant portion of cahced memoization table

void printTable(int A[][50], string str1, string str2, int m, int n)

{

//print out string two for top row of table

for (int a = 0; a < n; a++)

{

if (a == 0)

{

cout << setw(6) << str2.at(a); // first char starts 6 chars over

}

else

{

cout << setw(3) << str2.at(a); // rest separated by 3 spaces

}

}

cout << endl;

for (int i = 0; i <= m; i++) // num of rows; len (str1)

{

if (i != m) // printing str 1 downwards

{

cout << setw(3) << str1.at(i);

}

if (i == m) // printing last row

{

cout << setw(6) << A[i][0];

for (int j = 1; j <= n; j++) // column

{

cout << setw(3) << A[i][j];

}

cout << endl;

}

else // non last rows

{

for (int j = 0; j <= n; j++)

{

cout << setw(3) << A[i][j];

}

cout << endl;

}

}

cout << endl;

}

//Backtracking recursion function to determine map decisions on the corresponding

//insert, delete, replace operations. Write decision to stack for output

void backtracking(int dp[50][50], int m, int n, stack<char>& decisionOperations)

{

//Base case: dp[m][n] = 0

if (m == 0 && n == 0)

{

return;

}

//Start at bottom right cell, working towards top left cell

//If value is diagonal cell is smaller or equal to values in other two cells and if

//this is same or 1 - value of current cell, TAKE DIAGONAL CELL

// diagonal <= top value AND diagonal <= left value AND diagonal == 1-cur value OR diagonal == cur value

if ((((dp[m - 1][n - 1] <= dp[m - 1][n]) && (dp[m - 1][n - 1] >= 0)) && (dp[m - 1][n - 1] <= dp[m][n - 1] && (dp[m - 1][n - 1] >= 0))) && ((dp[m - 1][n - 1] == (dp[m][n] - 1)) || dp[m - 1][n - 1] == dp[m][n]))

{

//If value of diagonal cell is 1 less than current cell, REPLACE operation; otherwise no operation

if (dp[m - 1][n - 1] == (dp[m][n] - 1))

{

//decisions[m-1] = 'r'; //r corresponds to 's' in sundays

decisionOperations.push('r');

}

else

{

decisionOperations.push(' ');

}

backtracking(dp, m - 1, n - 1, decisionOperations); // recurse with diagonal value

}

//Value in cell to left is smaller or equal to value of cell above current cell AND if this value is same or 1-value of current cell

//INSERTION

else if ((((dp[m][n - 1] <= dp[m - 1][n]) && (dp[m - 1][n] >= 0)) || dp[m][n - 1] == 0) && ((dp[m][n - 1] == dp[m][n] - 1) || dp[m][n - 1] == dp[m][n]))

{

//decisions[m] = 'i';

decisionOperations.push('i');

backtracking(dp, m, n - 1, decisionOperations);

}

//Take cell above; DELETE operation

else

{

//decisions[m - 1] = 'd';

decisionOperations.push('d');

backtracking(dp, m - 1, n, decisionOperations);

}

//return; //should not reach here

}

// Function to construct cache memoization table to find minimal editing operations

int editDistDP(string str1, string str2, int m, int n, stack<char>& decisionOperations)

{

// Create a table to store results of subproblems

int dp[50][50]; // assume max lengths of strings to be compared is 50

// Fill dp[][] in bottom up manner

for (int i = 0; i <= m; i++) // m is length of first string 7

{

for (int j = 0; j <= n; j++) // n is length of second string 8

{

// If first string is empty, only option is to

// INSERT all characters of second string

if (i == 0)

dp[i][j] = j; // Min. operations = j

// If second string is empty, only option is to

// remove (DELETE) all characters of second string

else if (j == 0)

dp[i][j] = i; // Min. operations = i

// If last characters are same, ignore last char

// and recur for remaining string

else if (str1[i - 1] == str2[j - 1])

dp[i][j] = dp[i - 1][j - 1]; //get diagonal value

// If last character are different, consider all

// possibilities and find minimum

else

// 1 + min(insert, delete, replace)

dp[i][j] = 1 + min(dp[i][j - 1], // Insert i

dp[i - 1][j], // Remove (deletion) d

dp[i - 1][j - 1]); // Replace r

}

}

printTable(dp, str1, str2, m, n); // print the memoization table

backtracking(dp, m, n, decisionOperations); // backtrack to determine editing operations

return dp[m][n]; // returns # of editing operations

}

// Driver program

int main()

{

//string str1 = "sundays";

//string str2 = "saturday";

string str1 = "";

string str2 = "";

//Prompt user to input the two string values that are to be compared

cout << "Edit Distance: Charlie Ang Version" << endl;

cout << "Compute Edit Distance and character operations (insert, delete, replace)" << endl;

cout << "to transform one string into another string" << endl;

cout << endl;

cout << "Enter initial string: ";

cin >> str1;

cout << "Enter target string: ";

cin >> str2;

cout << endl;

char copyStr1[101]; //copy of str1 size 100 max length with room for null

char copyStr2[101]; //copy of str2 size 100 max length with room for null

char operations[101]; //outputting edit operations

stack<char> decisionOperations; // stack to store the edit operations

cout << "Minimum edits required to convert " << str1 << " into " << str2 << " is: " << editDistDP(str1, str2, str1.length(), str2.length(), decisionOperations) << endl;

int i = 0;

while (!decisionOperations.empty()) //pop all elements off stack to print out edit operations

{

char edit = decisionOperations.top();

decisionOperations.pop();

operations[i] = edit;

i++;

}

operations[i] = 0;

int indexStr1 = 0;

for (int i = 0; i < sizeof(operations); i++) //formatting str1 based on operations

{

char editOperation = operations[i];

if (editOperation == '\0') //null sentinel

{

break;

}

if (editOperation == 'i')

{

copyStr1[i] = ' ';

}

else

{

copyStr1[i] = str1.at(indexStr1);

indexStr1++;

}

}

copyStr1[i] = '\0';

int indexStr2 = 0;

//formatting str2 based on operations

for (int i = 0; i < sizeof(operations); i++) //copying string 2 into temp c-string

{

char editOperation = operations[i];

if (editOperation == '\0') //reached null sentinel

{

break;

}

if (editOperation == 'd') //output space if delete operation

{

copyStr2[i] = ' ';

}

else

{

copyStr2[i] = str2.at(indexStr2);

indexStr2++;

}

}

copyStr2[i] = '\0'; // null terminator

cout << right << setw(6) << "From: " << copyStr1 << endl;

cout << right << setw(6) << "Map: " << operations << "\t\t i)nsert d)elete r)eplace" << endl;

cout << right << setw(6) << "To: " << copyStr2 << endl;

return 0;

}